

METHOD AND DEVICE FOR DATA PROCESSING

Description

The present invention relates to what is claimed in the preamble and thus also relates to improvements in the use of reconfigurable processor technologies for data processing.

5 With respect to the preferred design of logic cell fields, reference is made here to the XPP architecture and previously published patent applications as well as more recent patent applications by the present applicant, these documents being fully incorporated herewith for disclosure purposes. The
10 following documents should thus be mentioned in particular:
DE 44 16 881 A1, DE 197 81 412 A1, DE 197 81 483 A1,
DE 196 54 846 A1, DE 196 54 593 A1, DE 197 04 044.6 A1,
DE 198 80 129 A1, DE 198 61 088 A1, DE 199 80 312 A1,
PCT/DE 00/01869, DE 100 36 627 A1, DE 100 28 397 A1,
15 DE 101 10 530 A1, DE 101 11 014 A1, PCT/EP 00/10516,
EP 01 102 674 A1, DE 198 80 128 A1, DE 101 39 170 A1,
DE 198 09 640 A1, DE 199 26 538.0 A1, DE 100 50 442 A1, as
well as PCT/EP 02/02398, DE 102 40 000, DE 102 02 044,
DE 102 02 175, DE 101 29 237, DE 101 42 904, DE 101 35 210,
20 EP 01 129 923, PCT/EP 02/10084, DE 102 12 622, DE 102 36 271,
DE 102 12 621, EP 02 009 868, DE 102 36 272, DE 102 41 812,
DE 102 36 269, DE 102 43 322, EP 02 022 692, as well as
EP 02 001 331 and EP 02 027 277.

One problem in traditional approaches to reconfigurable
25 technologies is encountered when the data processing is performed primarily on a sequential CPU using a configurable data processing logic cell field or the like and/or when data processing involving a plurality of processing steps and/or

extensive processing steps to be performed sequentially is desired.

There are known approaches which are concerned with how data processing may be performed on both a CPU and a configurable
5 data processing logic cell field.

WO 00/49496 describes a method for executing a computer program using a processor which includes a configurable functional unit capable of executing reconfigurable instructions, whose effect is redefinable in runtime by
10 loading a configuration program, this method including the steps of selecting combinations of reconfigurable instructions, generating a particular configuration program for each combination, and executing the computer program. Each time an instruction from one of the combinations is needed
15 during execution and the configurable functional unit is not configured using the configuration program for this combination, the configuration program for all the instructions of the combination is to be loaded into the configurable functional unit. In addition, a data processing
20 device having a configurable functional unit is known from WO 02/50665 A1, where the configurable functional unit is used to execute instructions according to a configurable function. The configurable functional unit has a plurality of independent configurable logic blocks for executing
25 programmable logic operations to implement the configurable function. Configurable connecting circuits are provided between the configurable logic blocks and both the inputs and outputs of the configurable functional unit. This allows optimization of the distribution of logic functions over the
30 configurable logic blocks.

One problem with traditional architectures occurs when coupling is to be performed and/or technologies such as data

streaming, hyperthreading, multithreading and so forth are to be utilized in a logical and performance-enhancing manner. A description of an architecture is given in "Exploiting Choice: Instruction Fetch and Issue on Implementable Simultaneous Multi-Threading Processor," Dean N. Tulson, Susan J. Eggers et al., Proceedings of the 23rd Annual International Symposium on Computer Architecture, Philadelphia, May 1996.

Hyperthreading and multithreading technologies have been developed in view of the fact that modern microprocessors gain their efficiency from many specialized functional units and functional units triggered like a deep pipeline as well as high memory hierarchies; this allows high frequencies in the function cores. However, due to the strictly hierarchical memory arrangements, there are major disadvantages in the event of faulty access to caches because of the difference between core frequencies and memory frequencies, since many core cycles may elapse before data is read out of the memory. Furthermore, problems occur with branchings and in particular incorrectly predicted branchings. It has therefore been proposed that a switch be performed between different tasks as a simultaneous multithreading procedure SMT whenever an instruction is not executable or does not use all functional units.

The technology of the above-cited exemplary documents (not by the present applicant) involves among other things an arrangement in which configurations are loadable into a configurable data processing logic cell field, but in which data exchange between the ALU of the CPU and the configurable data processing logic cell field, whether an FPGA, DSP or the like, takes place via registers. In other words, data from a data stream must first be written sequentially into registers and then stored in these registers sequentially again. Another problem occurs when there is to be external access to data,

because even then there are still problems in the
chronological data processing sequence in comparison with the
ALU and in the allocation of configurations, and so forth.
Traditional arrangements, such as those known from protective
5 rights not held by the present applicant are used, among other
things, for processing functions in the configurable data
processing logic cell field, DFP, FPGA or the like, which are
not efficiently processable on the ALU of the CPU. The
configurable data processing logic cell field is thus used in
10 practical terms to permit user-defined opcodes which allow
more efficient processing of algorithms than would be possible
on the ALU arithmetic unit of the CPU without configurable
data processing logic cell field support.

In the related art, as has been recognized, coupling is thus
15 usually word-based but not block-based, as would be necessary
for data streaming processing. It is initially desirable to
permit more efficient data processing than would be the case
with close coupling via registers.

Another possibility for using logic cell fields of logic cells
20 having a coarse and/or fine granular structure and logic cells
and logic cell elements having a coarse and/or fine granular
structure involves a very loose coupling of such a field to a
traditional CPU and/or a CPU core with embedded systems. A
traditional sequential program may run on a CPU or the like,
25 e.g., a program written in C, C++ or the like, data stream
processing calls being instantiated by this program on the
finely and/or coarsely granular data processing logic cell
field. It is then problematic that in programming for this
logic cell field, a program not written in C or another
30 sequential high-level language must be provided for data
stream processing. It would be desirable here for C programs
or the like to be processable on both the traditional CPU
architecture and on a data processing logic cell field

operated jointly together with [it], i.e., a data streaming capability is nevertheless maintained in quasi-sequential program processing using the data processing logic cell field in particular, whereas CPU operation in particular using a coupling which is not too loose remains possible at the same time. It is also already known that within a data processing logic cell field system such as that known in particular from PACT02 (DE 196 51 075.9-53, WO 98/26356), PACT04 (DE 196 54 846.2-53, WO 98/29952), PACT08 (DE 197 04 728.9, WO 98/35299), PACT13 (DE 199 26 538.0, WO 00/77652), PACT31 (DE 102 12 621.6-53, PCT/EP 02/10572), sequential data processing may also be provided within the data processing logic cell field. However, for example to save resources, to achieve time optimization and so forth, partial processing is achieved within a single configuration without this resulting in a programmer being able to automatically and easily implement a piece of high-level language code on a data processing logic cell field, as is the case with traditional machine models for sequential processors. Implementation of high-level language code on data processing logic cell fields according to the models for sequentially operating machines still remains difficult.

It is also known from the related art that multiple configurations, each triggering a different mode of functioning of array parts, may be processed simultaneously on the processor array (PA) and that a switch in one or more configurations may take place without any disturbance in others during runtime. Methods and means for their implementation in hardware are known; processing of partial configurations to be loaded into the field may be performed without a deadlock. Reference is made here in particular to the patent applications pertaining to the FILMO technology, e.g., PACT05 (DE 196 54 593.5-53, WO 98/31102), PACT10

(DE 198 07 872.2, WO 99/44147, WO 99/44120), PACT13
(DE 199 26 538.0, WO 00/77652), PACT17 (DE 100 28 397.7),
WO 02/13000); PACT31 (DE 102 12 621.6, WO 03/036507). This
technology already permits parallelization to a certain extent
5 and, with appropriate design and allocation of the
configurations, also permits a type of
multitasking/multithreading of such a type that planning,
i.e., scheduling and/or time use planning control, is
provided. Time use planning control means and methods are thus
10 known per se from the related art, allowing multitasking
and/or multithreading at least with appropriate allocation of
configurations to individual tasks and/or threads to
configurations and/or configuration sequences. The use of such
time use planning control means which have been used in the
15 related art for configuration and/or for configuration
management for the purpose of scheduling tasks, threads,
multithreads, and hyperthreads is regarded as inventive per
se.

It is also desirable, at least according to a partial aspect
20 in preferred variants, to be able to support modern
technologies of data processing and program processing such as
multitasking, multithreading, and hyperthreading, at least in
preferred variants of a semiconductor architecture.

The basic idea of the present invention is to provide a novel
25 device for commercial application.

This object is achieved by the method claimed in an
independent form. Preferred embodiments are described in the
subclaims.

A first essential aspect of the present invention may thus be
30 regarded as data being supplied to the data processing logic
cell field in response to execution of a load configuration by
the data processing logic cell field and/or data from this

data processing logic cell field is written back (STORED) by processing a STORE configuration accordingly. These load configurations and/or memory configurations are preferably to be designed in such a way that addresses of memory locations to be accessed directly or indirectly by loading and/or storage are generated directly or indirectly within the data processing logic cell field. Through this configuration of address generators within a configuration, a plurality of data is loadable into the data processing logic cell field, where it may be stored in internal memories (iRAM), if necessary, and/or in internal cells such as EALUs having registers and/or internal memory means. The load configuration and/or memory configuration thus allows loading of data by blocks, almost like datastreaming, in particular being comparatively rapid in comparison with individual access, and such a load configuration is executable before one or more configurations which process data by actually analyzing and/or modifying it, with which configuration(s) the previously loaded data is processed. Data loading and/or writing may typically take place in small areas of large logic cell fields, while other subareas are involved in other tasks. Reference is made to Figure 1 for these and other particulars of the present invention. In the ping-pong-like data processing described in other published documents by the present applicant in which memory cells are provided on both sides of the data processing field, one memory side may be preloaded with new data by a LOAD configuration in an array part, while data from the opposite memory side having a STORE configuration is written back in another array part; in a first processing step, data from the memory on one side streams through the data processing field to the memory on the other side, intermediate results obtained in the first stream through the field being stored in the second memory, the field being reconfigured, if necessary, and the interim results then streaming back for

further processing, etc. This simultaneous LOAD/STORE procedure is also possible without any spatial separation of memory areas.

It should be pointed out again that there are various possibilities for filling internal memories with data. The internal memories may be preloaded in advance in particular by separate load configurations using data streaming-like access. This would correspond to use as vector registers, resulting in the internal memories always being at least partially a part of the externally visible state of the XPP and therefore having to be saved, i.e., written back when there is a context switch. Alternatively and/or additionally, the internal memories (iRAMs) may be loaded onto the CPU through separate "load instructions." This results in reduced load processes through configurations and may result in a broader interface to the memory hierarchy. Here again, access is like access to vector registers.

Preloading may also include a burst from the memory through instruction of the cache controller. Moreover it is possible - and this is preferred as particularly efficient in many cases - to design the cache in such a way that a certain preload instruction maps a certain memory area, which is defined by the starting address and size and/or increment(s) onto the internal memory (iRAM). If all internal RAMs have been allocated, the next configuration may be activated. Activation entails waiting until all burst-like load operations are concluded. However, this is transparent if preload instructions are output long enough in advance and cache localization is not destroyed by interrupts or a task switch. A "preload clean" instruction may then be used in particular, preventing data from being loaded out of memory.

A synchronization instruction is needed to ensure that the content of a specific memory area stored cache-like in iRAM may be written back to the memory hierarchy, which may be accomplished globally or by specifying the accessed memory area; global access corresponds to a "full write-back." To simplify preloading of the iRAM, it is possible to specify this by simply giving a basic address, optionally one or more increments (in the event of access to multidimensional data fields) and a total run length, and to store this in registers or the like and then access these registers for determining how loading is to be performed.

It is particularly preferable for registers to be designed as FIFOs. One FIFO may then also be provided for each of a plurality of virtual processors in a multithreading environment. Moreover, memory locations may be provided for use as TAG memories, as is customary with caches.

It should also be pointed out that marking the content of iRAMS as "dirty" in the cache sense is helpful, so that the contents may be written back to an external memory as quickly as possible if the contents are not to be used again in the same iRAM. Thus the XPP field and the cache controller may be considered as a single unit because they do not need different instruction streams. Instead the cache controller may be regarded as the implementation of the steps "configuration fetch," "operand fetch" (iRAM preload) and "write-back," i.e., CF, OF and WB, in the XPP pipeline, the execution stage (ex) also being triggered. Due to the long latencies and unpredictability, e.g., due to faulty access to the cache or configurations of different lengths, it is advantageous if the steps are overlapped for the width of multiple configurations, the configuration and data preloading FIFO (pipeline) being used for the purpose of loose coupling. It should be pointed out that the FILMO, which is known per se, may be situated

downstream from the preload. It should also be pointed out that preloading may be speculative, the measure of speculation being determined as a function of the compiler. However, there is no disadvantage in incorrect preloading inasmuch as
5 configurations which have only been preloaded but have not been executed are readily releasable for overwriting, just as is the assigned data. Preloading of FIFO may take place several configurations in advance and may depend, for example, on the properties of the algorithm. It is also possible to use
10 hardware for this purpose.

With regard to writing back data used from iRAM to external memories, this may be accomplished by a suitable cache controller allocated to the XPP, but it should be pointed out that in this case, it will typically prioritize its tasks and
15 will preferentially execute preload operations having a high priority because of the assigned execution status. However, preloading may also be blocked by a higher-level iRAM instance in another block or by a lack of empty iRAM instances in the target iRAM block. In the latter case, the configuration may
20 wait until a configuration and/or a write-back is concluded. The iRAM instance in a different block may then be in use or may be "dirty." It is possible to provide for the clean iRAMs used last to be discarded, i.e., to be regarded as "empty." If there are neither empty nor clean iRAM instances, then a
25 "dirty" iRAM part and/or a nonempty iRAM part must be written back to the memory hierarchy. Only one instance may be in use at one time, and there should be more than one instance in an iRAM block to achieve a cache effect, so it is impossible that there are neither empty nor clean nor dirty iRAM instances.

30 Figures 4a through c illustrate examples of architectures in which an SMT processor is coupled to an XPP thread resource.

Even with the preferred variant presented here, it may be necessary to limit the memory traffic, which is possible in various ways during a context switch. For example, strict read data need not be stored, as is the case with configurations, for example. In the case of uninterruptible (non-preemptive) configurations, the local states of buses and PAEs need not be stored.

It is possible to provide for only modified data to be stored, and cache strategies may be used to reduce memory traffic. To do so, an LRU strategy (LRU = least recently used) may be implemented in particular in addition to a preload mechanism, in particular when there are frequent context switches.

If iRAMs are defined as local cache copies of the main memory and a starting address and modification state information are assigned to each iRAM, it is preferable for the iRAM cells to be replicated, as is also the case for SMT support, so that only the starting addresses of the iRAMs need be stored and loaded again as context. The starting addresses for the iRAMs of an instantaneous configuration then select the iRAM instances having identical addresses for use. If no address TAG of an iRAM instance corresponds to the address of the newly loaded context or the context to be newly loaded, the corresponding memory area may be loaded into an empty iRAM instance, this being understood here as a free iRAM area. If no such area is available, it is possible to use the methods described above.

Moreover, it should also be pointed out that delays caused by write-backs are avoidable by using a separate state machine (cache controller), with which an attempt is made in particular to write back iRAM instances which are inactive at the moment during unneeded memory cycles.

It should be pointed out that, as is apparent from the preceding discussion, the cache is preferably to be interpreted as an explicit cache and not as a cache which is transparent to the programmer and/or compiler as is usually the case. To provide the proper triggering here, the following instructions may be output, e.g., by the compiler: configuration preload instructions, which precede iRAM preload instructions which are used by that configuration. Such configuration preload instructions should be provided by the scheduler as soon as possible. Furthermore, i.e., alternatively and/or additionally, iRAM preload instructions which should likewise be provided by the scheduler at an early point in time may also be provided, and configuration execution instructions which follow iRAM preload instructions for this configuration may also be provided, these configuration execution instructions optionally being delayed, in particular by estimated latency times, in comparison with the preload instructions.

It is also possible to provide for a configuration wait instruction to be executed, followed by an instruction which orders a cache write-back, both being output by the compiler, in particular when an instruction of another functional unit such as the load/memory unit is able to access a memory area which is potentially dirty or in use in an iRAM. Synchronization of the instruction flows and cache contents may thus be forced while avoiding data hazards. Through appropriate handling, such synchronization instructions are not necessarily common.

It should be pointed out that data loading and/or storing need not necessarily take place in a procedure which is entirely based on logic cell fields. Instead it is also possible to provide one or more separate and/or dedicated DMA units, i.e., DMA controllers in particular, which are configured, i.e.,

functionally prepared, i.e., set up, e.g., by specifications with regard to starting address, increment, block size, target addresses, etc., in particular by the CT and/or from the logic cell field.

5 Loading may also be performed from and into a cache in particular. This has the advantage that external communication with larger memory banks is handled via the cache controller without having to provide separate switching arrangements within the data processing logic cell field; read or write
10 access in the case of cache memory means is typically very fast and has a low latency time; and typically a CPU unit is also connected to this cache, typically via a separate LOAD/STORE unit, so that access to data and exchange thereof by blocks may take place quickly between the CPU core and data
15 processing logic cell field, so that a separate command need not be fetched from the opcode fetcher of the CPU and processed for each transfer of data.

This cache coupling has also proven to be much more favorable than coupling of a data processing logic cell field to the ALU
20 via registers if these registers communicate with a cache only via a LOAD/STORE unit, as is known per se from the non-PACT publications cited above.

Another data link to the load/memory unit of the or one sequential CPU unit assigned to the data processing logic cell
25 field and/or to its registers may be provided.

It should be pointed out that such units may respond via separate input/output terminals (IO ports) of the data processing logic cell array designable in particular as a VPU and/or XPP and/or through one or more multiplexers downstream
30 from a single port.

It should also be pointed out that, in addition to blockwise and/or streaming and/or random reading and/or writing access, in particular in RMW mode (read-modify-write mode) to cache areas and/or the LOAD/STORE unit and/or the connection (known
5 per se in the related art) to the register of the sequential CPU, there may also be a connection to an external bulk memory such as a RAM, a hard drive and/or another data exchange port such as an antenna, etc. A separate port may be provided for this access to cache means and/or LOAD/STORE unit means and/or
10 memory means different from register units. It should be pointed out that suitable drivers, buffers, signal processors for level adjusting and so forth may be provided, e.g., LS74244, LS74245. It should also be pointed out that the logic cells of the field may include ALUs and/or EALUs, in
15 particular but not exclusively for processing a data stream flowing in or into the data processing logic cell field, and typically short fine-granularly configurable FPGA type circuits may be provided upstream from them at the inlet and/or outlet ends, in particular at both the inlet and outlet
20 ends, and/or may be integrated into the PAE-ALU to cut bit blocks out of a continuous data stream, for example, as is necessary for MPEG4 decoding. This is advantageous when a data stream is to enter the cell and is to be subjected there to a type of preprocessing without blocking larger PAEs units of
25 this type. This is also of particular advantage when the ALU is designed as a SIMD arithmetic unit, in which case a very long data input word having a data length of 32 bits, for example, is then split up via the upstream FPGA-type strips into a plurality of parallel data words having a length of 4
30 bits, for example, which may then be processed in parallel in the SIMD arithmetic units, which is capable of significantly increasing the overall performance of the system, if corresponding application[s] are needed. It should be pointed out that FPGA-type upstream and/or downstream structures were

discussed above. However, it should be pointed out explicitly that FPGA-type does not necessarily refer to 1-bit granular arrangements. It is possible in particular to provide, instead of these hyperfine granular structures, only fine granular structures having a width of 4 bits, for example. In other words, FPGA-type input and/or output structures upstream and/or downstream from an ALU unit designed as a SIMD arithmetic unit in particular are configurable, for example, so that 4-bit data words are always supplied and/or processed. It is possible to provide cascading here so that, for example, the incoming 32-bit-long data words stream into four separate and/or separating 8-bit FPGA-type structures positioned side by side, a second strip having eight 4-bit-wide FPGA-type structures is downstream from these four 8-bit-wide FPGA-type structures and then, if necessary, after another such strip, if necessary for the particular purpose, sixteen parallel 2-bit wide FPGA-type structures are also provided side by side, for example. If this is the case, a substantial reduction in configuration complexity may be achieved in comparison with strictly hyperfine granular FPGA-type structures. It should also be pointed out that this also results in the configuration memory of the FPGA-type structure possibly turning out to be much smaller, thus permitting a savings in terms of chip area. It should also be pointed out that FPGA-type strip structures, as also shown in conjunction with Figure 3, in particular situated in the PAE, permit implementation of pseudo-random noise generators in a particularly simple manner. If individual output bits obtained stepwise always from a single FPGA cell are written back to the FPGA cell, a pseudo-random noise may also be generated creatively using a single cell, which is considered to be inventive per se (see Figure 5).

In principle, the coupling advantages in the case of data block streams described above are achievable via the cache, but it is particularly preferable if the cache is designed in slices and then multiple slices are simultaneously accessible, in particular all slices being simultaneously accessible. This is advantageous when a plurality of threads is to be processed on the data processing logic cell field (XPP) and/or the sequential CPU(s), as explained below, whether via hyperthreading, multitasking and/or multithreading. Cache memory means having slice access and/or slice access enabling control means are therefore preferably provided. For example, a separate slice may be assigned to each thread. This makes it possible later in processing the threads to ensure that the proper cache areas are accessed when the command group to be processed using the thread is resumed.

It should be mentioned again that the cache need not necessarily be divided into slices, and if this is the case, a separate thread need not necessarily be assigned to each slice. However, it should be pointed out that this is by far the preferred method. It should also be pointed out that there may be cases in which not all cache areas are being used simultaneously or temporarily at a given point in time. Instead it is to be expected that in typical data processing applications such as those occurring with handheld mobile telephone (cell phones), laptops, cameras and so forth, there are frequently times during which the entire cache is not needed. It is therefore particularly preferable if individual cache areas are separable from the power supply so that their power consumption drops significantly, in particular to zero or almost zero. In a slice-wise cache design, this may occur by shutting down the cache in slices via suitable power disconnection means (see Figure 2, for example). The disconnection may be accomplished either by cycling down,

clock disconnection, or power disconnection. In particular, access recognition may be assigned to an individual cache slice or the like, this access recognition being designed to recognize whether a particular cache area, i.e., a particular
5 cache slice, has a thread, hyperthread, or task assigned to it at the moment, by which it is being used. If the access recognition means then ascertains that this is not the case, typically disconnection from the clock and/or even from the power will then be possible. It should be pointed out that on
10 reconnecting the power after a disconnection, immediate response of the cache area is possible again, i.e., no significant delay need be expected due to turning the power supply on and off if implemented in hardware using conventional suitable semiconductor technologies. This is
15 appropriate in many applications independently of the use with logic cell fields.

Another particular advantage obtained with the present invention is that although there is particularly efficient coupling with respect to the transfer of data and/or operands
20 in blockwise form in particular, nevertheless no balancing is necessary in such a way that exactly the same processing time is necessary in a sequential CPU and XPP and/or data processing logic cell field. Instead, the processing is performed in a manner which is practically often independent,
25 in particular in such a way that the sequential CPU and the data processing logic cell field system may be considered as separate resources for a scheduler or the like. This allows immediate implementation of known data processing program splitting technologies, such as multitasking, multithreading,
30 and hyperthreading. The resulting advantage that path balancing is not necessary, i.e., balancing between sequential parts (e.g., on a RISC unit) and data flow parts (e.g., on an XPP), results in any number of pipeline stages optionally

being run through, e.g., within the sequential CPU (i.e., the RISC functional units), for example, cycling in a different way is possible and so forth. Another advantage of the present invention is that by configuring a load configuration and/or a store configuration into the XPP or other data processing logic cell fields, the data may be loaded into the field or written out of it at a rate which is no longer determined by the clock speed of the CPU, the speed at which the opcode fetcher works or the like. In other words, the sequence control of the sequential CPU is no longer a bottleneck restriction for the data throughput through the data cell logic field [**sic; data processing logic cell field**] without there being even a loose coupling.

In a particularly preferred variant of the present invention, it is possible to use known CTs (or CMs = configuration managers or configuration tables) for an XPP unit to use the configuration of one or more XPP fields also designed hierarchically with multiple CTs and at the same time one or more sequential CPUs more or less as multithreading scheduler and hardware management, which has the inherent advantage that known technologies (FILMO, etc.) may be used for the hardware-supported management in multithreading, but alternatively and/or additionally, in particular in a hierarchical arrangement, it is possible for a data processing logic cell field like an XPP to receive configurations from the opcode fetcher of a sequential CPU via the coprocessor interface. This results in a call being instantiatable by the sequential CPU and/or another XPP, resulting in data processing on the XPP. The XPP is then kept in the data exchange, e.g., via the cache coupling described here and/or via LOAD and/or STORE configurations which provide address generators for loading and/or write-back of data in the XPP and/or data processing logic cell field. In other words, coupling of a data

processing logic cell field in the manner of a coprocessor and/or thread resources is possible while at the same time data loading in the manner of data streaming is taking place through cache coupling and/or I/O port coupling.

5 It should be pointed out that the coprocessor coupling, i.e., the coupling of the data processing logic cell field, will typically result in scheduling for this logic cell field as well as also taking place on the sequential CPU or on a higher level scheduler unit and/or corresponding scheduler means. In
10 such a case, threading control and management takes place in practical terms on the scheduler and/or the sequential CPU. Although this is possible per se, this will not necessarily be the case at least in the simplest implementation of the present invention. Instead, the data processing logic cell
15 field may be used by calling in the traditional way as is done with a standard coprocessor, e.g., in the case of 8086/8087 combinations.

In addition, it should be pointed out that in a particularly preferred variant, regardless of the type of configuration,
20 whether via the coprocessor interface, the configuration manager (CT) of the XPP and/or of the data processing logic cell field or the like, where the CT also functions as a scheduler, or in some other way, it is possible, in and/or directly on the data processing logic cell field and/or under
25 management of the data processing logic cell field, to address memories, in particular internal memories, in particular in the case of the XPP architecture, such as that known from the various previous patent applications and publications by the present applicant, RAM PAEs or other similarly managed or
30 internal memories as a vector register, i.e., to store the data quantities loaded via the LOAD configuration like vectors as in vector registers in the internal memories and then after reconfiguring the XPP and/or the data processing logic field,

i.e., overwriting and/or reloading and/or activating a new configuration which performs the actual processing (in this context, it should be pointed out that for a such a processing configuration, reference may also be made to a plurality of configurations which are to be processed in wave mode and/or sequentially), to access them as in the case of a vector register and then store the results thus obtained and/or intermediate results in turn in the internal memories or external memories managed via the XPP like internal memories to store these results there. The memory means written in this way in the manner of a vector register with processing results using XPP access are then written back in a suitable manner by loading the STORE configuration after reconfiguring the processing configuration; this in turn takes place in the manner of data streaming, whether via the I/O port directly into external memory areas and/or, as is particularly preferred, into cache memory areas which may then be accessed by the sequential CPU and/or other configurations on the XPP, which previously generated the data or another corresponding data processing unit, may access.

According to a particularly preferred variant, at least for certain data processing results and/or interim results, the memory means and/or vector register means in which the resulting data is to be stored is not an internal memory into which data may be written via STORE configuration in the cache area or some other area which the sequential CPU or another data processing unit may access, but instead the results are written directly into corresponding cache areas, in particular access-reserved cache areas, which may be organized like slices in particular. This may have the disadvantage of a greater latency, in particular when the paths between the XPP or data processing logic cell field unit and the cache are so long that the signal propagation times become significant, but

it may result in no additional STORE configuration being needed. It should also be pointed out that such storage of data in cache areas is possible, as described above, due to the fact that the memory to which the data is written is
5 located in physical proximity to the cache controller and is designed as a cache, but alternatively and/or additionally there is also the possibility of placing part of an XPP memory area, XPP-internal memory or the like, in particular in the case of RAM via PAEs (see PACT31: DE 102 12 621.6, WO
10 03/036507), under the management of one or more sequential cache memory controllers. This has advantages when minimizing the latency when storing the processing results, which are determined within the data processing logic cell field, whereas the latency in the case of access by other units to
15 the memory area, which then functions only as a "quasi-cache," plays little or no role.

It should also be pointed out that, according to another embodiment, the cache controller of the traditional sequential CPU addresses a memory area as a cache, this memory area being
20 physically located on and/or at the data processing logic cell field without being used for the data exchange with it. This has the advantage that, when applications having a low local memory demand are running on the data processing logic cell field, and/or when only a few additional configurations are
25 needed, based on the available storage volume, this may be available as a cache to one or more sequential CPUs. It should be pointed out that the cache controller may then be and will be designed for management of a cache area having a dynamic extent, i.e., of varying size. Dynamic cache size management
30 and/or cache size management means for dynamic cache management will typically take into account the work load and/or the input/output load on the sequential CPU and/or the data processing logic cell field. In other words, it is

possible to analyze, for example, how many NOPs data accesses there are in a given unit of time to the sequential CPU and/or how many configurations in the XPP field should be stored in advance in memory areas provided for this purpose to be able
5 to permit rapid reconfiguration, whether by way of wave reconfiguration or by some other means. The dynamic cache size described here is thus particularly preferably a runtime dynamic, i.e., the cache controller manages a prevailing cache size, which may change from one clock pulse to the other or
10 from one clock pulse group to the other. Moreover, it should be pointed out that the access management of an XPP and/or data process logic cell field including access as an internal memory as is the case with a vector register and as a cache-type memory for external access, with regard to the memory
15 accesses, has already been described in DE 196 54 595 and PCT/DE 97/03013 (PACT03). The publications cited are herewith incorporated fully by reference thereto for disclosure purposes.

Reference was made above to data processing logic cell fields
20 which are runtime reconfigurable in particular. The fact that a configuration management unit (CT and/or CM) may be provided for these systems was discussed. Management of configurations per se is known from the various patents and applications by the present applicant, to which reference has been made for
25 disclosure purposes, as well as the applicant's other publications. It shall now be pointed out explicitly that such units and their mechanism of operation via which configurations not yet currently needed are preloadable, in particular independently of connections to sequential CPUs,
30 etc., are also highly usable for inducing a task switch and/or a thread switch and/or a hyperthread switch in multitasking operation and/or in hyperthreading and/or in multithreading (see **[Figures]** 6a through 6c, for example). To do so, it is

possible to utilize the fact that, during the runtime of a thread or task, configurations for different tasks, i.e., threads and/or hyperthreads, may also be loaded into the configuration memory in the case of a single cell or a group of cells of the data processing logic cell field, i.e., a PAE of a PAE field (PA), for example. As a result, in the case of a blockade of a task or thread, e.g., when it is necessary to wait for data because the data is not yet available, whether because it has not yet been generated or received by another unit, e.g., because of latencies, or because a resource is currently still being blocked by another access, then configurations for another task or thread are preloadable and/or preloaded and it is possible to switch to them without the time overhead of having to wait for a configuration switch in the case of a shadow-loaded configuration in particular. In principle, it is possible to use this technique even when the most probable continuation is predicted within a task and a prediction is not correct (prediction miss), but this type of operation is preferred in prediction-free operation. In the case of use with a purely sequential CPU and/or multiple purely sequential CPUs, in particular exclusively with such CPUs, multithreading management hardware is thus implemented by adding a configuration manager. Reference is made in this regard in particular to PACT10 (DE 198 07 872.2, WO 99/44147, WO 99/44120) and PACT17 (DE 100 28 397.7, WO 02/13000). It may be regarded as sufficient, in particular if hyperthreading management is desired for a CPU and/or a few sequential CPUs, to omit certain partial circuits like the FILMO as described in the patents and applications to which reference has been made specifically. In particular, this also describes the use of the configuration manager described there with and/or without FILMO for hyperthreading management for one or more purely sequentially operating CPUs with or without connection to an XPP or another data processing logic cell field and is

herewith claimed separately. A separate and particular inventive feature is seen herein. It should also be pointed out that a plurality of CPUs may be implemented using the known techniques, as are known in particular from PACT31

5 (DE 102 12 621.6-53, PCT/EP 02/10572) and PACT34 (DE 102 41 812.8, PCT/EP 03/09957) in which one or more sequential CPUs are provided within an array, utilizing one or more memory areas in the data processing logic cell field in particular for construction of the sequential CPU, in
10 particular as an instruction register and/or data register. It should also be pointed out here that previous patent applications such as PACT02 (DE 196 51 075.9-53, WO 98/26356), PACT04 (DE 196 54 846.2-53, WO 98/29952), and PACT08 (DE 197 04 728.9, WO 98/35299) have already disclosed how
15 sequencers having ring and/or random access memories may be constructed.

It should be pointed out that a task switch and/or a thread switch and/or a hyperthread switch using the known CT technology – see PACT10 (DE 198 07 872.2, WO 99/44147,
20 WO 99/44120) and PACT17 (DE 100 28 397.7, WO 02/13000) – may take place and preferably will take place; that performance slices and/or time slices are assigned by the CT to a software-implemented operating system scheduler or the like which is known per se, during which it is determined which
25 parts per se are to be processed subsequently by which tasks or threads, assuming that resources are free. An example may be given in this regard as follows: first, an address sequence is to be generated for a first task; according to this, data is to be loaded from a memory and/or cache memory to which a
30 data processing logic cell field is connected in the manner described here, during the execution of a LOAD configuration. As soon as this data is available, processing of a second data processing configuration, i.e., the actual data processing

configuration, may be initiated. This may also be preloaded because it is certain that this configuration is to be executed as long as no interrupts or the like require a complete task switch. In traditional processors, there is the
5 problem known as cache miss, in which data is requested but is not available in the cache for load access. If such a case occurs in a coupling according to the present invention, it is possible to switch preferably to another thread, hyperthread and/or task which was intended for the next possible execution
10 in particular by the operating system scheduler implemented through software in particular and/or another similarly acting unit, and therefore was loaded, preferably in advance, into one of the available configuration memories of the data processing logic cell field, in particular in the background
15 during the execution of another configuration, e.g., the LOAD configuration which has triggered the loading of the data for which the system is now waiting. It should also be mentioned here explicitly that separate configuration lines may lead from the configuring unit to the particular cells directly
20 and/or via suitable bus systems, such as those known in the related art per se, for advance configuration, undisturbed by the actual wiring of the data processing logic cells of the data processing logic cell field having a close granular design in particular, because this design is particularly
25 preferred here to permit undisturbed advance configuration without interfering with another configuration underway at that moment. Reference is to be made here to PACT10 (DE 198 07 872.2, WO 99/44147, WO 99/44120), PACT17 (DE 100 28 397.7, WO 02/13000), PACT13 (DE 199 26 538.0, WO 00/77652), PACT02 (DE 196 51 075.9, WO 98/26356) and PACT08 (DE 197 04 728.9, WO 98/35299). If the configuration to which the system has switched during and/or because of the task thread switch and/or hyperthread switch has been processed and processing has been completed in the event of preferably

indivisible, uninterruptible and thus quasi-atomic configurations - see PACT19 (DE 102 02 044.2, WO 2003/060747) and PACT11 (DE 101 39 170.6, WO 03/017095) - then in some cases another configuration is processed as predetermined by the corresponding scheduler, in particular the scheduler close to the operating system and/or the configuration for which the particular LOAD configuration was executed previously. Before execution of a processing configuration for which a LOAD configuration has previously been executed, it is possible to test in particular, e.g., by query of the status of the load configuration or the data loading DMA controller, to determine whether in the meantime the particular data has streamed into the array, i.e., whether the latency time has elapsed, as typically occurs, and whether the data is actually available.

In other words, if latency times occur, e.g., because configurations have not yet been configured into the system, data has not yet been loaded and/or data has not yet been written back, they will be bridged and/or masked by the execution of threads, hyperthreads and/or tasks which have already been preconfigured and are operating using data which is already available and/or which may be written back to resources which are already available for write-back. Latency times are largely covered in this way and virtually 100% utilization of the data processing logic cell field is achieved, assuming an adequate number of threads, hyperthreads and/or tasks to be executed per se.

It should be pointed out in particular that by providing an adequate number of XPP-internal memory resources which are freely assigned to threads, e.g., by the scheduler or the CT, the cache and/or write operations of several simultaneous and/or superimposed threads may be executed, which has a particularly positive effect on bridging any latencies.

Using the system described here with regard to data stream capability in the case of simultaneous coupling to a sequential CPU and/or with regard to coupling an XPP array and/or data processing logic cell field and simultaneously a sequential CPU to a suitable scheduler unit such as a configuration manager or the like, real time-capable systems are readily implementable in particular. For real time capability, it is necessary to ensure a response to incoming data and/or interrupts signaling the arrival of data in particular within a maximum period of time, which is not to be exceeded in any case. This may be accomplished, for example, by a task switch to an interrupt and/or, e.g., in the case of prioritized interrupts, by ascertaining that a given interrupt is to be ignored at the moment, in which case this must also be defined within a certain period of time. A task switch in such real time-capable systems is typically achievable in three ways, namely either when a task has been running for a certain period of time (timer principle), when a resource is not available, whether due to being blocked by some other access or due to latencies in access thereto, in particular reading and/or writing access, i.e., in the case of latencies in data access, and/or in the event of occurrence of interrupts.

It is also pointed out that a runtime-limited configuration in particular may also trigger a watchdog and/or parallel counter on a resource which is to be enabled and/or switched for processing the interrupt.

Although it has otherwise been stated explicitly - see also PACT29 (DE 102 12 622.4, WO 03/081454) - that new triggering of the parallel counter and/or watchdog to increase runtime is suppressible by a task switch, according to the present invention, an interrupt may also have a blocking effect, i.e., according to a task switch, parallel counter - and/or watchdog

- and new trigger, i.e., in such a case it is possible to prevent the configuration itself from increasing its maximum possible runtime by new triggering.

According to the present invention, the real time capability
5 of a data processing logic cell field may now be achieved by implementing one or more of three possible variants.

According to a first variant, within a resource addressable by the scheduler and/or the CT, there is a switch to processing an interrupt, for example. If the response times to interrupts
10 or other requests are so long that a configuration may still be processed without interruption during this period of time, then this is noncritical in particular, since a configuration for interrupt processing may be preloaded onto the resource which is to be switched to processing the interrupt, and this
15 may be done during processing of the currently running configuration. The choice of the interrupt processing configuration to be preloaded is to be made by the CT, for example. It is possible to limit the runtime of the configuration on the resource which is to be enabled and/or
20 switched for the interrupt processing. Reference is made in this regard to PACT29/PCT (PCT/DE03/000942).

In systems which must respond to interrupts more quickly, it may be preferable to reserve a single resource, i.e., for example, a separate XPP unit and/or parts of an XPP field for
25 such processing. If an interrupt which must be processed quickly then occurs, it is possible to either process a configuration preloaded for particularly critical interrupts in advance or to begin immediately loading an interrupt processing configuration into the reserved resource. A choice
30 of the particular configuration required for the corresponding interrupt is possible through appropriate triggering, wave processing, etc.

It should also be pointed out that using the methods already described, it is readily possible to obtain an instant response to an interrupt by achieving code re-entrance by using LOAD/STORE configurations. After each data processing configuration or at given points in time, e.g., every five or ten configurations, a STORE configuration is executed and then a LOAD configuration is executed while accessing the memory areas to which data was previously written. When it is certain that the memory areas used by the STORE configuration will remain unaffected until another configuration has stored all relevant information (states, data) by progressing in the task, it is then certain that the same conditions will be obtained again on reloading, i.e., on re-entrance into a configuration previously initiated but not completed. Such an insertion of LOAD/STORE configurations with simultaneous protection of STORE memory areas which are not yet outdated is very easily generated automatically without additional programming complexity, e.g., by a compiler. Resource reservation may be advantageous there. It should also be pointed out that in resource reservation and/or in other cases, it is possible to respond to at least a quantity of highly prioritized interrupts by preloading certain configurations.

According to another particularly preferred variant of the response to interrupts, when at least one of the addressable resources is a sequential CPU, an interrupt routine in which a code for the data processing logic cell field is prohibited is to be processed on it. In other words, a time-critical interrupt routine is processed exclusively on a sequential CPU without calling XPP data processing steps. This ensures that the processing operation on the data processing logic cell field is not to be interrupted and then further processing may take place on this data processing logic cell field after a

task switch. Although the actual interrupt routine does not have an XPP code, it is nevertheless possible to ensure that at a later point in time, which is no longer relevant to real time, following an interrupt it is possible to respond with the XPP to a state and/or data detected by an interrupt and/or a real time request using the data processing logic cell field.